



TITLE:

不完全指定形抽象データタイプの仕様記述と実現 (計算の複雑性に関する研究)

AUTHOR(S):

坂部, 俊樹; 稲垣, 康善; 本多, 波雄

CITATION:

坂部, 俊樹 ...[et al]. 不完全指定形抽象データタイプの仕様記述と実現 (計算の複雑性に関する研究). 数理解析研究所講究録 1980, 381: 258-275

ISSUE DATE:

1980-04

URL:

<http://hdl.handle.net/2433/104807>

RIGHT:

不完全指定形抽象データタイプの仕様記述と実現

坂部俊樹[†] 稲垣康善^{††} 本多波雄[†]

([†]名大・工 ^{††}三重大・工)

1 まえがき 抽象データタイプは、信頼性が高く効率的でありかつ柔軟性のあるソフトウェアの開発には欠かせない重要な概念として注目され、特に、その形式的仕様記述を手える代数的手法について多くの研究がなされている[1~7]。代数的仕様記述の基本的な考え方は、抽象データタイプを *sort* 代数として捉え、公理(等式)の有限集合で与えられる仕様によって記述される抽象データタイプは公理を満たす代数のカテゴリの始代数であるとするものである。従来、このような議論の中で取り扱われる代数の演算はすべて全域的に定義される関数であった。そのために、抽象データタイプの演算のうち未定義部分があってもよい演算も、UNDEFINED または ERROR という特別な値を用いて、全域的な関数として取り扱われ、そこに多少の不自然さが見うけられた。そこで本報告では、まずはじめに演算が一般に部分関数であるような

抽象データタイプ (これを不完全指定形抽象データタイプと呼ぶ) を導入して, その代数的仕様記述の方法を明らかにする。この方法によれば, 実際には起こり得ない演算の組合せ等を未定義部分のある演算で表現することにより, 多くの場合に従来の仕様記述と比べて簡潔な仕様記述が得られる。さらに, 不完全指定形抽象データタイプの演算の未定義部分を効果的に利用する実現を定式化し, 従来のモデルでしばしば見られたむだな条件判断 (IF THEN ELSE 演算の重複) を自然に除去できるなど, より効率の良い実現が得られることも示す。

2 部分関数を演算とする多 sort 代数 この章では, 本報告で中心的役割を果たす演算が部分関数である多 sort 代数およびそのカテゴリについて簡単に説明する。本報告では, Goguen 他 [1] の流儀に沿って議論が進められるので, なるべく [1] と共通の記法を用いるものとし, それらについての詳しい定義等は省略することがある。

sort の集合を S , S -sorted operator domain を $\Sigma = \langle \Sigma_{w,A} \rangle_{w \in S^*, A \in S}$ とする。演算の合成を表す sort A の Σ 項の集合を $T_{\Sigma,A}$ と書き, 変数の入っている sort A の Σ 項 ($\Sigma(X)$ 項と呼ぶ) の集合を $T_{\Sigma}(X)_A$ と書く。ただし X は変数集合族 $\langle X_A \rangle_{A \in S}$ である。 $t \in T_{\Sigma}(X)_A$ の中の変数が x_1, \dots, x_n のとき, これを明確にするために $t(x_1, \dots, x_n)$ と書くことがある。また, t を木で表すこと

ができるので、木の深さは木の高さで定義する。 Σ 代数 A は集合族 $\langle A_\alpha \rangle_{\alpha \in S}$ と各演算記号 $\sigma \in \Sigma_{w, A}$ ($w = \alpha_1 \cdots \alpha_n$) に対する部分関数 $\sigma_A: A_{\alpha_1} \times A_{\alpha_2} \times \cdots \times A_{\alpha_n} \rightarrow A_\sigma$ とからなる。ただし、 $\sigma \in \Sigma_{\lambda, A}$ (λ は空系列) に対しては、 $\sigma_A \in A_\lambda$ とする。注意すべき点は、[1, 3, 7] 等で使われる代数と異なり、演算が部分関数となっていることである。これに伴ない、合成演算を表す $\Sigma(X)$ 項木の A 上での解釈 t_A (木によって定められる A 上の合成演算 (derived operation) という。) も部分関数となる。すなわち、 t_A の定め方は関数の引数のどれか一つでも未定義ならその関数は未定義とする通常の部分関数合成の定義に基づく。

Σ 代数のカテゴリは Σ 代数間の準同形写像を定めることによって与えられる。すべての演算が全域的である Σ 代数に対する準同形写像の定義をそのまま流用して、演算が部分関数である Σ 代数の準同形写像を定義し、それを Σ 準同形写像 と呼ぶ。すなわち、 Σ 代数 A, B に対して、次の条件 (h0), (h1) を満たす写像族 $h = \langle h_\alpha: A_\alpha \rightarrow B_\alpha \rangle_{\alpha \in S}$ を Σ 準同形写像 といい、 $h: A \rightarrow B$ と書く。

$$(h0) \quad \sigma \in \Sigma_{\lambda, A} \text{ ならば } h_\sigma(\sigma_A) = \sigma_B.$$

$$(h1) \quad \sigma \in \Sigma_{\alpha_1 \cdots \alpha_n, A} \text{ ならば } h_\sigma \circ \sigma_A = \sigma_B \circ (h_{\alpha_1} \times \cdots \times h_{\alpha_n}).$$

ただし、 \circ は部分関数の合成であり、写像 $h_{\alpha_1} \times \cdots \times h_{\alpha_n}: A_{\alpha_1} \times \cdots \times A_{\alpha_n} \rightarrow B_{\alpha_1} \times \cdots \times B_{\alpha_n}$ は $h_{\alpha_1} \times \cdots \times h_{\alpha_n}(a_1, \dots, a_n) = (h_{\alpha_1}(a_1), \dots, h_{\alpha_n}(a_n))$ で定めら

れる. Σ 準同形写像 $h: A \rightarrow B$ と $h': B \rightarrow C$ の合成 $h' \circ h: A \rightarrow C$ を $\langle h'_\alpha \circ h_\alpha \rangle_{\alpha \in S}$ で定義する.

このとき次の命題が得られる. すべての Σ 代数のクラスを Alg_Σ , すべての Σ 準同形写像のクラスを \mathcal{H}_Σ (あるいは単に \mathcal{H}) とするとき,

命題 2.1 $(Alg_\Sigma, \mathcal{H}_\Sigma)$ はカテゴリである. 以下では, このカテゴリを \mathcal{H}_Σ で表すことにする.

Σ 準同形写像は不完全指定形抽象データタイプを導入して自由度のある実現を考察するという我々の目標にとってはやや制限が強いので, Σ 準同形写像より制限の弱い準同形写像を次に定義する.

任意の非負整数を n とする. 以下では非負整数の集合を ω と書く. Σ 代数 A, B に対して, 条件 (h0) および次の条件 (h2), (h3) を満たす写像の族 $h = \langle h_\alpha: A_\alpha \rightarrow B_\alpha \rangle_{\alpha \in S}$ を n 準同形写像 という.

(h2) $\sigma \in \Sigma_{\alpha_1, \dots, \alpha_n, \alpha}$ ならば $h_\alpha \circ \sigma_A \subseteq \sigma_B \circ (h_{\alpha_1} \times \dots \times h_{\alpha_n})$.

(h3) $t(x_1, \dots, x_n) \in T_\Sigma(X)_\alpha$, $x_i \in X_{\alpha_i}$ かつ $|t| \leq n$ ならば,

$h_\alpha \circ t_A \supseteq (h_\alpha(A_\alpha) \upharpoonright t_B \upharpoonright h_{\alpha_1}(A_{\alpha_1}) \times \dots \times h_{\alpha_n}(A_{\alpha_n})) \circ (h_{\alpha_1} \times \dots \times h_{\alpha_n})$. ただし, $h_\alpha(A_\alpha) \upharpoonright t_B \upharpoonright h_{\alpha_1}(A_{\alpha_1}) \times \dots \times h_{\alpha_n}(A_{\alpha_n})$ は部分関数 $t_B: B_{\alpha_1} \times \dots \times B_{\alpha_n} \rightarrow B_\alpha$ の定義域を $h_{\alpha_i}(A_{\alpha_i}) = \{h_{\alpha_i}(a) \mid a \in A_{\alpha_i}\} (\subseteq B_{\alpha_i})$ の直積へ制限し, 値域を $h_\alpha(A_\alpha) (\subseteq B_\alpha)$ へ制限した部分関数である.

すべての n 準同形写像のクラスを $n\text{-}\mathcal{H}_\Sigma$ (あるいは単に $n\text{-}\mathcal{H}$) で表す. 特に, 0 準同形写像, すなわち (h0) と (h2) を満たす $\langle h_\alpha \rangle_{\alpha \in S}$ を 弱準同形写像 と呼ぶことがある. また, 任意の $n \in \omega$ に対して (h3) が成り立つ弱準同形写像を ∞ 準同形写像という. さらに, すべての $\alpha \in S$ に対して $h_\alpha: A_\alpha \rightarrow B_\alpha$ が全(単)射である n 準同形写像 h を n 全(単)準同形写像 といい, そのクラスを $n\text{-}\mathcal{H}_\Sigma^{\text{onto}}$ ($n\text{-}\mathcal{H}_\Sigma^{1:1}$) で表す.

このように定義される n 準同形写像について次の性質がある.

性質 2.1 適当に Σ を選ぶと, 任意の $n \in \omega$ に対して

$$n\text{-}\mathcal{H}_\Sigma^{1:1} \supseteq (n+1)\text{-}\mathcal{H}_\Sigma^{1:1} \supseteq \infty\text{-}\mathcal{H}_\Sigma^{1:1} \quad (\supseteq \text{ は真の包含関係})$$

性質 2.2 任意の Σ に対して

$$1\text{-}\mathcal{H}_\Sigma^{\text{onto}} = 2\text{-}\mathcal{H}_\Sigma^{\text{onto}} = \dots = \infty\text{-}\mathcal{H}_\Sigma^{\text{onto}}$$

性質 2.3 適当に Σ を選ぶと, $0\text{-}\mathcal{H}_\Sigma \supseteq \mathcal{H}_\Sigma \supseteq 1\text{-}\mathcal{H}_\Sigma^{\text{onto}}$

性質 2.4 $0\text{-}\mathcal{H}_\Sigma$, $n\text{-}\mathcal{H}_\Sigma^{1:1}$ ($n \in \omega$), $\infty\text{-}\mathcal{H}_\Sigma^{1:1}$, $0\text{-}\mathcal{H}_\Sigma^{\text{onto}}$, $1\text{-}\mathcal{H}_\Sigma^{\text{onto}}$ は合成のもとで閉じている.

性質 2.5 $n\text{-}\mathcal{H}_\Sigma$ ($n \geq 1$) および $\infty\text{-}\mathcal{H}_\Sigma$ は合成のもとで閉じていない.

性質 2.4, 2.5 から本報告で新たに定めた n 準同形写像の一部のクラスだけが Σ 代数のカテゴリを形成することが知られる. すなわち,

命題 2.2 $0\text{-}\mathcal{H}_\Sigma$, $n\text{-}\mathcal{H}_\Sigma^{1:1}$ ($n \in \omega$), $\infty\text{-}\mathcal{H}_\Sigma^{1:1}$, $0\text{-}\mathcal{H}_\Sigma^{\text{onto}}$, $1\text{-}\mathcal{H}_\Sigma^{\text{onto}}$ は

カテゴリである。

3 不完全指定形抽象データタイプの仕様記述 本章では、まず、不完全指定形抽象データタイプの仕様記述の形式的定義を手え、そのあとで、抽象データタイプの典型的な例である *Stack* の仕様記述を例としてとりあげる。尚以下では、特に断わらないう限り、仕様記述と言えは不完全指定形の仕様記述を意味する。

Σ を *S-sorted-operator domain* とする。

定義 3.1 Σ 等式 は対 $e = (u, v)$ である。ただし、 $\Sigma(x)$ 項 u , v は同一の sort である。以下では e を $u = v$ と書く。 Σ 代数 A は $u_A = v_A$ である時かつその時に限り e を 満足する という。 Σ 等式の集合 E に対して、 A がすべての E の元を満足する時かつその時に限り A は E を満足するという。

従来の完全指定形の仕様記述は、この Σ 等式の有限集合で定義されていたが、このままでは演算の定義域に関する情報が不足で、不完全指定形抽象データタイプの仕様記述としては不十分である。そこで次の定義を手える。

定義 3.2 Σ 代数を A とする。 Σ 項 $t \in T_{\Sigma, A}$ は、 $t_A \in A_s$ の時かつその時に限り A で 定義される という。 Σ 項の集合 $\Delta = \bigcup_{s \in S} \Delta_s$ ($\Delta_s \subseteq T_{\Sigma, A}$) は Δ のすべての元が A で定義されるときかつその時に限り A で定義されるという。

定義 3.3 不完全指定形抽象データタイプの仕様記述は 3 項組 (Σ, E, Δ) である. ここに, Σ は S -sorted operator domain, E は Σ 等式の有限集合, Δ は Σ 項の集合である.

さて, このように定義された仕様記述の意味, すなわち, 仕様記述によって記述される不完全指定形抽象データタイプを次のように定める.

定義 3.4 仕様記述 (Σ, E, Δ) によって記述される不完全指定形抽象データタイプは, E を満足しかつ Δ が定義されるすべての Σ 代数のクラスによって定められる $0\text{-}\mathcal{H}_\Sigma$ の部分カテゴリ $0\text{-}\mathcal{H}_{\Sigma, E, \Delta}$ の始代数である.

仕様記述の意味を直観的に理解するため, $0\text{-}\mathcal{H}_{\Sigma, E, \Delta}$ の始代数の具体的な構成を手える. そのために次の定義を手える.

定義 3.5 Σ 代数 A 上の Σ 合同 \equiv は次の条件を満たす各 $\alpha \in S$ に対する A_α 上の同値関係 \equiv_α からなる族 $\langle \equiv_\alpha \rangle_{\alpha \in S}$ である.

$\sigma \in \Sigma_{\alpha_1, \dots, \alpha_n, \alpha}$, $a_i \equiv_{\alpha_i} a'_i$ ($i=1, \dots, n$) かつ $\sigma_A a = (a_1, \dots, a_n)$ に定義されるならば $\sigma_A a$ は (a'_1, \dots, a'_n) にも定義され, かつ, $\sigma_A (a_1, \dots, a_n) \equiv_\alpha \sigma_A (a'_1, \dots, a'_n)$. $a \in A_\alpha$ の \equiv_α による同値類を $[a]$ と書く. Σ 合同 \equiv に関する A の商代数 A/\equiv は通常のように定義される. すなわち, (i) 各 $\alpha \in S$ に対して $(A/\equiv)_\alpha = A_\alpha / \equiv_\alpha$. (ii) 各 $\sigma \in \Sigma$ に対して, (i) $\sigma \in \Sigma_{\alpha_1, \dots, \alpha_n, \alpha}$ ならば $\sigma_{A/\equiv} = [\sigma_A]$, (ii) $\sigma \in \Sigma_{\alpha_1, \dots, \alpha_n, \alpha}$ ならば, $[a_i] \in (A/\equiv)_{\alpha_i}$ ($i=1, \dots, n$) に対して,

$$\sigma_{A/\equiv}([a_1], \dots, [a_n]) = \begin{cases} [\sigma_A(a_1, \dots, a_n)] & ; \sigma_A \text{ が } (a_1, \dots, a_n) \text{ に定義される時,} \\ \text{未定義 ; その他} \end{cases}$$

定義 3.6 $\langle T_{\Sigma, A} \rangle_{A \in S}$ 上で各演算を次のように定めて得られる Σ 代数を T_Σ とする。各々の $\sigma \in \Sigma$ に対して,

- (i) $\sigma \in \Sigma_{\lambda, A}$ ならば $\sigma_{T_\Sigma} = \sigma (\in T_{\Sigma, A})$
- (ii) $\sigma \in \Sigma_{A_1 \dots A_n, A}$ かつ $t_i \in T_{\Sigma, A_i}$ ならば $\sigma_{T_\Sigma}(t_1, \dots, t_n) = \sigma(t_1, \dots, t_n)$

定義 3.7 Σ 等式の集合 E によって生成される Σ 代数 T_Σ 上の Σ 合同を \equiv とする。ここで, (Σ, E) を従来の完全指定形の仕様記述とみなすと, それによつて記述される完全指定形抽象データタイプは T_Σ / \equiv (以下では $T[\Sigma, E]$ と書く) であることに注意しよう。

さて, 仕様記述 (Σ, E, Δ) に対して次のような Σ 代数 $U[\Sigma, E, \Delta]$ (あるいは単に U) を構成する。各 $A \in S$ に対して, 集合 $U_A \subseteq T[\Sigma, E]_A (= T_{\Sigma, A} / \equiv_A)$ を

$$U_A = \{ [a] \in T_{\Sigma, A} / \equiv_A \mid \exists A' \in S, \exists t(x) \in T_\Sigma(X)_{A'}, \exists b \in \Delta_{A'} \cup \Sigma_{\lambda, A'}, x \in X_{A'} \text{ かつ } t(a) \equiv_{A'} b \}$$

で定める。各演算記号 σ に対して,

- (1) $\sigma \in \Sigma_{\lambda, A}$ ならば $\sigma_U = [\sigma]$,
- (2) $\sigma \in \Sigma_{A_1 \dots A_n, A}$ かつ $[a_i] \in U_{A_i} \ (i=1, \dots, n)$ ならば,

$$\sigma_U([a_1], \dots, [a_n]) = \begin{cases} [\sigma(a_1, \dots, a_n)] & ; [\sigma(a_1, \dots, a_n)] \in U_A \\ \text{未定義 ; その他} \end{cases}$$

と定める。この定義が正当で \mathcal{U} が Σ 代数であることは容易に確かめられる。構成から明らかのように、 \mathcal{U} の $\text{sort } \mathcal{A}$ の台 $U_{\mathcal{A}}$ は、定数もしくは Δ の元 b と \equiv 等価な Σ 項 $t(a)$ を表す木の部分木に対応する Σ 項 a の \equiv 同値類 $[a]$ からなる。 $\sigma \in \Sigma_{A_1, \dots, A_n, A}$ に対する演算 $\sigma_{\mathcal{U}} : U_{A_1} \times \dots \times U_{A_n} \rightarrow U_A$ は、全域的な演算 $\sigma_{T[\Sigma, E]} : (T_{\Sigma, A_1} / \equiv_{A_1}) \times \dots \times (T_{\Sigma, A_n} / \equiv_{A_n}) \rightarrow (T_{\Sigma, A} / \equiv_A)$ の定義域 $U_{A_1} \times \dots \times U_{A_n}$ 値域 U_A への制限であり必ずしも全域的ではない。

この \mathcal{U} に因りて次の命題が成立つ。

命題 3.1 任意の仕様記述 (Σ, E, Δ) に対して、 $\mathcal{U}[\Sigma, E, \Delta]$ は $\mathcal{O}\text{-}\mathcal{H}_{\Sigma}[E, \Delta]$ の始代数である。

さて、このように定められた不完全指定形抽象データタイプの仕様記述と従来の完全指定形の仕様記述との違いを、典型的な抽象データタイプである *Stack* を例にと、て見てみよう。ここで考える *Stack* の演算は *NEWSTACK*, *PUSH*, *POP*, *TOP*, *ISNEW* であり、それらの意味は [1~4] 等で見られるものと同じとする。ただし、*NEWSTACK* に対して、*POP* および *TOP* は未定義であるとする。

次ページの図 3.1 および図 3.2 は、それぞれ、不完全指定形および完全指定形の立場からの *Stack* の仕様記述である。尚、完全指定形の仕様記述では、未定義部分のある演算を取り扱うために特別な値 *UNDEFINED* を用いている。

不完全指定形と完全指定形の仕様記述の最も大きな差は等式集合にみられる。例えばこの例では、完全指定形の仕様記述の等式集合から演算の未定義性に関わる等式を除いたものが不完全指定形仕様記述の等式集合になっている。

さらに厳密に言えば、完全指定形仕様記述においてはずべての等式を、*ok-propagation*, *error-propagation* [1] の考え方に従って変形

し、*Stack* の本質とは関係しないような余分の等式を加えなければならぬ。これに対して、不完全指定形の仕様記述の等式集合は *Stack* の本質を表すと思われる等式のみからなり簡潔である。

もう一つの差は、不完全指定形の場合には Δ の指定が必要であるのに対して完全指定形ではそれを必要としない。従っ

```

type Stack[Element]
syntax ( $\Sigma$ )
  NEWSTACK :  $\rightarrow$  Stack
  PUSH : Stack  $\times$  Element  $\rightarrow$  Stack
  POP : Stack  $\rightarrow$  Stack
  TOP : Stack  $\rightarrow$  Element
  ISNEW : Stack  $\rightarrow$  Bool
semantics ( $E$ )
  (1) ISNEW(NEWSTACK) = TRUE
  (2) ISNEW(PUSH(s,e)) = FALSE
  (3) POP(PUSH(s,e)) = s
  (4) TOP(PUSH(s,e)) = e
defined terms ( $\Delta$ )
   $\phi$ 

```

図3.1 *Stack* の不完全指定形仕様記述

```

type Stack[Element]
syntax
  NEWSTACK :  $\rightarrow$  Stack
  PUSH : Stack  $\times$  Element  $\rightarrow$  Stack
  POP : Stack  $\rightarrow$  Stack
  TOP : Stack  $\rightarrow$  Element
  ISNEW : Stack  $\rightarrow$  Bool
semantics
  (1) ISNEW(NEWSTACK) = TRUE
  (2) ISNEW(PUSH(s,e)) = FALSE
  (3) POP(PUSH(s,e)) = s
  (4) POP(NEWSTACK) = UNDEFINED
  (5) TOP(PUSH(s,e)) = e
  (6) TOP(NEWSTACK) = UNDEFINED

```

図3.2 *Stack* の完全指定形仕様記述

てこの点に因しては完全指定形の方が簡潔であるといえる。
 しかし、実際の抽象データタイプの多くは、Stackの例でもそ
 うであるように、どのデータ対象からも適当な合成演算によ
 ってある定数に到達できるというある意味での連結性を持
 つているので、 $\Delta = \phi$ であることが多い。また、 ϕ でなくとも
 Δ の表現は多くの場合簡単である。

以上のように、不完全指定形の仕様記述は、従来の完全指
 定形の仕様記述と形式的にも、また、その意味のとりえ方に
 おいても基本的には同じで、しかも、演算の未定義性の取り
 扱いの煩雑さを軽減している。しかしながら次のような問題
 点がある。上のStackの場合には使用されなかったが、一般に
 は IF THEN ELSE 演算の入った等式がよく用いられる。IF THEN ELSE
 (x, a, b) は $x = \text{TRUE}$ のときは a 、 $x = \text{FALSE}$ のときは b となる演
 算であり、IF THEN ELSE と他の部分関数との合成は、

$$\begin{aligned} & \text{IF THEN ELSE} (f(x_1, \dots, x_n), g(y_1, \dots, y_m), h(z_1, \dots, z_n)) \\ &= \begin{cases} g(y_1, \dots, y_m); f, g, h \text{ が定義されかつ } f(x_1, \dots, x_n) = \text{TRUE} \\ h(z_1, \dots, z_n); f, g, h \text{ が定義されかつ } f(x_1, \dots, x_n) = \text{FALSE} \\ \text{未定義; その他} \end{cases} \end{aligned}$$

で定義される。この定義の考え方は、プログラムの不動点理
 論や完全指定形仕様記述に見られる、未定義要素上や ERROR
 要素による未定義性の取扱いに必要な拡張された IF THEN ELSE

の定義の考え方は異なる。従って、IF THEN ELSEの入った等式から直観的に意味を捉える際にその等式をプログラムとみなして理解すると仕様記述の意味を誤って理解する恐れがある。不完全指定形の仕様記述の中で IF THEN ELSE を用いるときには引数の値が定義されるかあるいは未定義であるかに細心の注意が必要である。

4 不完全指定形抽象データタイプのn実現 不完全指定形抽象データタイプの実現は、従来の抽象データタイプの実現と同様に、*deriver* の概念を用いて定式化される。不完全指定形抽象データタイプの演算は部分関数なので、演算の未定義部分をどのように実現するかという点に注意が必要である。

本報告で定式化されるn実現(n は非負整数)は次のような考えに基づいている。今、仕様記述によって与えられた抽象データタイプ A, B, C があり、 A の一部を B で、また、 B の一部を C で実現することを考える。このとき、 A の一部を B で実現すると主に使われる情報は A および B の仕様記述だけであり、 B が C により、どのように実現されているかという情報は使用されない(してはいけない)。さらに、 B による A の実現を考えると主には、 B の仕様記述から未定義になるような B 上の演算の組合せはわかっているので、定義されるど

のような A の演算の組合せも決して未定義である B の演算の組合せに対応しないようにする。このようにすれば、 B の演算の未定義部分が C によってどのように実現されても A の演算の定義される部分には影響がないようにできる。換言すると、 A の演算の定義される部分だけに注目する限り、 B の演算を C の合成演算で実現するとすれば B の演算の未定義部分と自由に実現してもかまわない。このことは、丁度、組合せ禁止（すなわち *don't care*）のある論理関数の実現の自由度があることに類似している。

さて、以上の考え方に従って不完全指定形抽象データタイプの n 実現を定式化しよう。まず *derivator* の定義を与える。

定義 4.1 Σ を S -sorted operator domain, Ω を S' -sorted operator domain とする。 Σ から Ω への *derivator* d は、写像 $f: S \rightarrow S'$ と写像族 $\langle d_{w,s}: \Sigma_{w,s} \rightarrow (T_{\Omega}(Y))_{f(w), f(s)} \rangle_{w \in S^*, s \in S}$ との対である。ただし、 f の定義域は自然に S^* へ拡張されるとし、 $(T_{\Omega}(Y))_{f(w), f(s)}$ ($w = s_1 \dots s_n$) は、 $y_i \in X_{f(s_i)}$ ($i = 1, \dots, n$) であるような sort $f(s)$ の $\Omega(Y)$ 項 $t(y_1, \dots, y_n)$ すべての集合である。 B を Ω 代数とすると、 d -derived 代数 dB は、 $(dB)_s = B_{f(s)}$ ($s \in S$)、かつ、若 $\sigma \in \Sigma_{w,s}$ に対して $\sigma_{dB} = (d_{w,s}(\sigma))_B$ で定義される Σ 代数である。

定義 4.2 n を非負整数とする。仕様記述 (Σ, E, Δ) の n 実現は 4 項組 (B, d, \equiv, h) である。ここに、 B は Ω 代数、 d

type Arrayint

syntax

NEWARRAY : \rightarrow Array

ASSIGN : Array \times Integer \times Element \rightarrow Array

ACCESS : Array \times Integer \rightarrow Element

\langle , \rangle : Array \times Integer \rightarrow Arrayint (pairing function)

P1 : Arrayint \rightarrow Array (projection function)

P2 : Arrayint \rightarrow Integer (projection function)

semantics

(1) ACCESS(ASSIGN(a,i,e),i) = e

(2) ASSIGN(ASSIGN(a,i,e),i',e')
= IF (i=i') THEN ASSIGN(a,i',e')
ELSE ASSIGN(ASSIGN(a,i',e'),i,e)

(3) P1(\langle a,i \rangle) = a

(4) P2(\langle a,i \rangle) = i

(5) \langle P1(b),P2(b) \rangle = b

defined terms

{ ASSIGN(...(ASSIGN(NEWARRAY,i1,e1)...),in,en) | n = 1,2,... }

図4.1 Arrayintの仕様記述

は Σ から Ω への derivor, \equiv は \mathcal{AB} 上の Σ 合同, \mathcal{A} は $\mathcal{U}[\Sigma, E, \Delta]$ から \mathcal{AB} への n 準準同形写像である。

この n 実現を直観的に理解するために, 配列と整数の直積の抽象データタイプ Arrayint による Stack の 1 準実現を例としてあげる。Stack は本章で与えられた仕様記述 (Σ, E, Δ) (図3.1) で定められる Σ 代数 $\mathcal{U}[\Sigma, E, \Delta]$ とする。抽象データタイプ Arrayint は図4.1の仕様記述で定められるものとする。Arrayint は S' -sorted Ω 代数であるとする。さて, Σ から Ω への derivor d

を図4.2のよう

に定める。次

に, d Arrayint 上

の Σ 合同 \equiv を

次の集合と含

f-part

Stack \mapsto Arrayint

Element \mapsto Element

Bool \mapsto Bool

d-part

NEWSTACK \mapsto \langle NEWARRAY, 0 \rangle

ISNEW \mapsto "P2(b)=0"

PUSH \mapsto \langle ASSIGN(P1(b), P2(b)+1, e), P2(b)+1 \rangle

POP \mapsto \langle P1(b), P2(b)-1 \rangle

TOP \mapsto ACCESS(P1(b), P2(b))

図4.2 Σ から Ω への derivor d

を最小の Σ 合同[†]とする。

$$\{ \langle a, i \rangle, \langle a', i' \rangle \mid i = i' \text{ かつ } \forall j \leq i, \text{ACCESS}(a, j) = \text{ACCESS}(a', j) \}$$

次に h は $d \equiv$ から自然に定められる写像族とする。このとき h は 1 準同形写像である。このことは、少々面倒であるが d および \equiv の定め方から示される。 h が 2 準同形写像ではないことは次のように示される。 $Stack$ の仕様記述から明らかのように、深さ 2 の合成演算 $PUSH(POP(a), e)$ が $Stack$ 上では $NEWSTACK$ に対して未定義であるが、 $dArrayint$ においては定義され $(PUSH(POP(NEWSTACK), e))_{dArrayint} \equiv NEWSTACK_{dArrayint}$ である。このことと、 $h(NEWSTACK_{Stack}) = NEWSTACK_{dArrayint} / \equiv$ であることとを合わせると $n=2$ の場合の条件 ($h3$) が満たされないことがわかる。従って、 $(Arrayint, d, \equiv, h)$ は $Stack$ の 1 実現である。

同じ $Stack$ を従来の完全指定形の立場から実現するとほとんど同様の実現が得られる。しかし POP 演算は、

$$(*) \quad \text{IF } P2(b)=0 \text{ THEN UNDEFINED ELSE } \langle P1(b), P2(b)-1 \rangle$$

という $Arrayint$ 上の合成演算に対応付けられ、上の 1 実現と比べると深さが大きくなっていて、効率の点では 1 実現の方が優れている。また、 $Stack$ を使って他の抽象データタイプを完全指定形の立場で実現するとき、UNDEFINED を故意に使う場合は例外として、 POP 演算の前には通常現在のスタックが空か否

† 関係 R を含む最小の Σ 合同は、一般には存在しない。

なとチェックする必要があるので、そのための IF THEN ELSE と上の(*)の IF THEN ELSE がひどな重複となる。ところが、不完全指定形抽象データタイプのn実現では d と \equiv とうまく工夫することによってそのような重複を避けることができる。

完全指定形の場合の実現と不完全指定形でのn実現とのような違いは、Stackの例に限らず一般的に見られるものであり、AをBで実現し、BをCで実現し、CをDで実現し、... というように実現の段数が大きくなればなるほどその差は大きくなると予想される。

5 あとがき 本報告では、従来の完全指定形の立場からの抽象データタイプの仕様記述の手法における演算の未定義性の取扱いの煩雑さを軽減するため、不完全指定形抽象データタイプを導入し、その仕様記述法を手えた。そしてこの仕様記述法によって得られる仕様記述は、完全指定形の立場からの仕様記述と比べて等式の数が少ないなどの利点があることも例でしめした。

さらに、不完全指定形の抽象データタイプの実現として、n実現を定式化し、n実現は従来の実現と比べて自由度が大きく、かつ、効率の点で優れていることを Arrayint による Stack の1実現の例で示した。

今後検討すべき課題として、3章で述べた IF THEN ELSE 演算

の取扱いに関する問題, また, それに関連して, λ 実現の recursion を許した場合に起こる問題, さらに,仕様記述と λ 実現の正当性の証明に関する問題などが考えられる.

本筆ながら, 御指導賜わる名古屋大学福村晃夫教授, 並びに, 討論して頂いた研究室の諸氏に深謝する.

文 献

- [1] J.A.Goguen, J.W.Thatcher and E.G.Wagner; An initial algebra approach to the specification, correctness and implementation of abstract data types, IBM Research Report, RC-6487 (1976).
- [2] J.V.Guttag, E.Horowitz and D.R.Musser; Abstract data types and software validation, CACM,21,12(1978).
- [3] 嵩,谷口,杉山,萩原,鈴木,奥井; プログラムの仕様記述の代数的手法について, 信学技報 AL78-5 (1978).
- [4] B.H.Liskov and S.N.Zilles; Specification techniques for data abstraction, IEEE Trans. on SE, SE-1, 1(1975).
- [5] 杉山,谷口,嵩; 代数的仕様記述に基づくプログラム設計の一例, 信学技報 AL79-13 (1979).
- [6] 鈴木,杉山,萩原,谷口,嵩,奥井; プログラムの仕様と実現化の代数的記述, 信学技報 AL78-46 (1978).
- [7] J.W.Thatcher, E.G.Wagner and J.B.Wright; Data type specification: Parameterization and the power of specification techniques, Proceedings SIGACT 10th Annual Symposium on Theory of Computing.

type Array[Integer,Element]

syntax ()

NEWARRAY : Array

ASSIGN : Array Integer Element Array

ACCESS : Array Integer Element

semantics ()

(1) ACCESS(ASSIGN(a,i,e),i) = e

(2) ASSIGN(ASSIGN(a,i,e),i',e')

= IF (i=i') THEN ASSIGN(a,i',e')

ELSE ASSIGN(ASSIGN(a,i',e'),i,e)

defined terms ()

ASSIGN(ASSIGN(...(ASSIGN(NEWARRAY,i1,e1),i2,e2)...),in,en)

n = 1,2,...